NetServ: Deploying Customized Network Services on Demand

Henning Schulzrinne,
Jae Woo Lee & Suman Srinivasan
Columbia University

Joint work with: Bell Labs (Alcatel-Lucent), Deutsche Telekom,
DOCOMO Euro-Labs

NetServ overview

Extensible architecture for core network services

- Modularization
  - Building Blocks
  - Service Modules

- Virtual services framework
  - Security
  - Portability

NSF FIND four-year project
  - Columbia University
  - Bell Labs
  - Deutsche Telekom
  - DOCOMO Euro-Labs

No more ossification in NGI
MIA Overview

All network elements may offer
- communication (global)
- computation
- storage

internetworking layer

everywhere fast & low cost

general-purpose CPU

language binding

network API

common functionality modules (e.g., pub-sub, CDN)
signaling (install state & code)

Network node example

storage & computation

multiple computation & storage providers

data center or POP
Different from Active Networks?

- **Active Networks**
  - Packet contains executable code
  - Can modify router states and behavior
  - Not successful
    - Per-packet processing too expensive
    - Security concerns
    - No compelling *killer app* to warrant such a big shift
  - Notable work: ANTS, Janos, Switchware

- **NetServ**
  - Virtualized services on current, passive networks
    - Service invocation is signaling driven, not packet driven
  - Service modules are stand-alone, addressable entities
    - Separate from packet forwarding plane
    - Extensible plug-in architecture

Building Blocks

- **Key components of network services**
  - Access to network-level resource
  - Implementation of common functionality

- **For example:**
  - Link monitoring and measurement
  - Routing table
  - Packet capture
  - Data storage and lookup
Service Modules

- Full-fledged service implementations
  - Use Building Blocks and other Service Modules
  - Can be implemented across multiple nodes
  - Invoked by applications
- Examples:
  - Routing-related services
    - Multicast, anycast, QoS-based routing
  - Monitoring services
    - Link & system status, network topology
  - Identity services
    - Naming, security
  - Traffic engineering services
    - CDN, redundancy elimination, p2p network support

Deployment Scenarios

- Three actors
  - Content publisher (e.g. youtube.com)
  - Service provider (e.g. ISP)
  - End user
- Model 1: Publisher-initiated deployment
  - Publisher rents router space from providers (or end users)
- Model 2: Provider-initiated deployment
  - Publisher writes NetServ module
  - Provider sees lots of traffic, fetches and installs module
  - Predetermined module location (similar to robots.txt)
- Model 3: User-initiated deployment
  - User installs NetServ module to own home router or PC
Where does code run?

- All (or some?) nodes in a network
  - AS, enterprise LAN
- Some or all nodes along path
  - data path from source to destination
- Selected nodes by property
  - e.g., one in each AS

How does code get into nodes?

All nodes in (enterprise) network
gossip
How does code get into nodes?

**NSIS**

- Progress along data path
  - with RAO-based discovery
- Designed to transport large objects
  - supports TCP and UDP
- Security mechanisms
First prototype implementation

- Proof-of-concept for dynamic network service deployment
  - Open-source Click modular router
  - Java OSGi dynamic module system
- Promising initial measurement results
  - NetServ overhead acceptable compared to other overhead

Technology: Click router

- Runs as a Linux kernel module or user-level program
- Modules written in C++ (called *Elements*) are configured in a text file
- Elements are arranged in a directed graph, through which packets traverse
- Example:
  - Click router command:
    ```bash
    sudo click print.click
    ```
  - Configuration file `print.click`:
    ```plaintext
    FromDevice(en0)->CheckIPHeader(14)->IPPrint->Discard;
    ```
- [http://www.read.cs.ucla.edu/click/](http://www.read.cs.ucla.edu/click/)
Technology: OSGi

- Dynamic module system for Java
  - Modules loaded and unloaded at runtime
  - Bundle: self-contained JAR file with specific structure
  - Open-source implementations: Apache Felix, Eclipse Equinox

- Security and accounting
  - Security built on Java 2 Security model
    - Permission-based access control
    - No fine-grained control or accounting for CPU, storage, bandwidth
    - Can load native code with appropriate permission
  - Strict separation of bundles
    - Classpath set up by Bundle class loader
    - Inter-bundle communication only through published interfaces

1st prototype implementation

- Implements PktProcessor
- Registers an instance of PktDispatchingService
- packet flow
- Single process

NetServ App Bundle
NetServ Building Block Bundle
Equinox OSGi framework
NetServ OSGi Launcher
Java Virtual Machine
StaticIP lookup element
NetServ element
CheckIPHeader element
User-level Click router
Demo: NetServ prototype

1. Regular incoming packets
2. "Operator" can view modules on router
3. Operator loads a new module (that makes all data uppercase)
4. Packets are modified
5. Operator stops the module
6. No more packet modification

Performance Evaluation

- Initial measurements on the first prototype
  - NetServ on user-level Click router
  - Maximum Loss Free Forward Rate (MLFFR)
- Future work on next-generation prototypes
  - NetServ on JUNOS, kernel-mode Click
  - Ping latency
  - Microbenchmarks
  - Throughput for non-trivial services
For a modular architecture, kernel-user transition is unavoidable since putting a module inside a kernel is not an option.

Current Work: CDN on NetServ

- On-Path CDN
  - Prototype implemented
- Dynamic content migration
  - Moving content closer to the end user according to demand
- Building blocks
  - Network monitoring
  - Content discovery
  - Caching proxy
API examples

• Avoid SNMP retrieval problems
  – all or nothing (typical)
  – hard to do selective triggers
• Flow management
  – counters, measurement
• System information
  – like system MIB: geo location, uptime, interface speeds, …
  – routing table
  – routing table changes (“tell me if route to X changes”)

Current Work: NetServ Platform

• Ubiquitous NetServ
  – From big to small devices
  – Real router: Juniper’s JUNOS
  – Personal computer: Kernel-mode Click
  – Home router: Linux using iptables
• Security and resource control
  – Enable various deployment scenarios
  – Support different economic incentives
Related Work

- Cisco’s Programmable Overlay Router
- Juniper’s JUNOS SDK
- DaVinci project
- VROOM (virtual routers on the move)
- OpenFlow Switch
- Ethane

Future Internet Architecture?

- Really closer to urban design
  - zoning, fire codes and infrastructure (rail, water)
    - plus oversight (fire marshal & building inspector)
  - architecture changes, urban designs stay
    - see Washington, DC & Berlin
- “Architecture” must be
  - expressible in one sentence
  - avoid limiting options (unknown unknowns)
  - avoid imposing unnecessary costs
The network services fallacy

• We tried adding network services as protocols:
  – multicast
  – QoS
  – mobility
  – security
• All were, more-or-less, failures
  – (or underperformed expectations)
  – hard to secure, not quite right

Thoughts on architecture

• Long-term constant: service model
  – equivalent of railroad track & road width
• Identify core functions we need
  – routing
  – congestion control
  – name lookup
  – path state establishment
  – …
• Learn from history
  – why didn’t these get done “right”?
• Need engineering principles
• Requirement list doesn’t help
MIA

• “Deliver packets from point A to B”
  – where A and B are globally unique identifiers

Summary

• NetServ: architecture for dynamic in-network service deployment
• Modular and extensible
  – Building Blocks and Service Modules
• Secure and portable
  – Virtualized Services Framework
• And it is NOT Active Networks
• Prototype implementation: Click and OSGi
• Supports various deployment scenarios
• CDN application under development